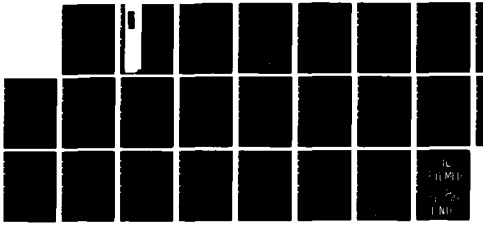
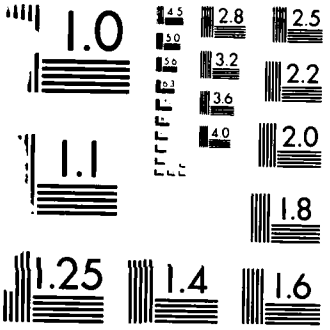
AD-A165 242 DESIGN AND IMPLEMENTATION OF DATA STRUCTURES FOR GENERALIZED NETWORKS(U) TEXAS UNIV AT AUSTIN CENTER FOR CYBERNETIC STUDIES I ALI ET AL. MAR 84 CCS-RR-483 1/1

UNCLASSIFIED N00014-81-C-0236 F/G 12/2 NL





(12)

Research Report CCS 483

DESIGN AND IMPLEMENTATION OF DATA STRUCTURES FOR GENERALIZED NETWORKS

by

I. Ali
A. Charnes
T. Song

CENTER FOR CYBERNETIC STUDIES
The University of Texas
Austin, Texas 78712

12

Research Report CCS 483

DESIGN AND IMPLEMENTATION OF DATA STRUCTURES FOR GENERALIZED NETWORKS

by

I. Ali
A. Charnes
T. Song

March 1984

This paper was partly supported by ONR Contract N00014-81-C-0236, ONR Contract N00014-82-K-0295 and USARI Contract MDA 903-85-M-6548. Reproduction in whole or in part is permitted for any purpose of the United States Government.

CENTER FOR CYBERNETIC STUDIES

A. Charnes, Director
Graduate School of Business 4.138
The University of Texas at Austin
Austin, Texas 78712
(512) 471-1821

ABSTRACT

The specialization of the simplex algorithm for the solution of generalized network flow problems rests on the fact that a basis for the problem may be represented graphically as a spanning forest in which each component is either a one-tree or a rooted tree. The design of a specialized algorithm for efficient solution of generalized network problems necessarily depends on data structures chosen to represent the basis. This paper presents the design and detailed algorithmic specification of the primal simplex algorithm for such problems. Computational testing to determine the overhead required by generalized network data structures over pure network data structures indicates that generalized network algorithms are on the order of 2.5 to 3.5 times slower than pure network algorithms. Computational testing with generalized network problems with up to 1000 nodes and 7000 arcs establishes the suitability of the data-structures for efficient implementation of primal simplex calculations.

Keywords:

Networks

Generalized Networks

Algorithms

I. Introduction

The generalized network problem is a specialized linear programming problem which may be defined as

$$\begin{aligned} \min \quad & cx \\ \text{s.t.} \quad & Ax = r \\ & 0 \leq x \leq u \end{aligned}$$

where A is a nxm node-arc incidence matrix with multipliers on the arcs. That is, each column, a_k, has at most two non-zero entries, d_k in row i and g_k in row j. The problem may be easily scaled so that one of the entries, say d_k is a + 1. If g_k is zero, then the variable corresponds to a loop. Otherwise, the column a_k represents an arc from node i to node j where d_k units leave node i, producing g_k units at node j. Without loss of generality, we assume that the matrix A corresponds to a connected network, G(N,E), where the rows correspond to the node set N of the network, and the variables correspond to the arc set E of the network. Further, we assume that the matrix A is of full row rank. It can be shown that if the rank is n-1, then the problem corresponds to a pure network problem [4].

The specialization of the simplex algorithm for the solution of such problems rests on the fact that the basis for a generalized network problem corresponds to a graphical structure referred to as a spanning forest. Much in the same manner that the development of specialized solution software for the pure network problem rests on data structures for the representation of a spanning tree, specialized solution software for solution of the generalized network problem rests on the representation of a spanning forest.

The design of a specialization of the simplex method for the solution of generalized network problems is necessarily dependent on the data structure

chosen to represent the basis. The basis for a generalized network flow problem has the following form:

$$\begin{bmatrix} B_1 & & & \\ & B_2 & & \\ & & \ddots & \\ & & & B_t \end{bmatrix}$$

where each of the t submatrices B_i has the form:

$$\begin{bmatrix} d_1 & & & & & \\ g_1 & d_2 & & & & \\ & g_2 & d_3 & & & \\ & & \ddots & \ddots & & \\ & & & d_p & & \\ & & & & \ddots & \\ & & & & d_{q-1} & \\ & & & & g_{q-1} & d_q \end{bmatrix}$$

(a) 1-tree with self loop

or

$$\begin{bmatrix} d_1 & & & & & \\ g_1 & d_2 & & & & \\ & g_2 & d_3 & & & \\ & & \ddots & \ddots & & \\ & & & d_p & & g_q \\ & & & g_{p-1} & \ddots & \\ & & & & d_{q-1} & \\ & & & & g_{q-1} & d_q \end{bmatrix}$$

(b) 1-tree with loop

The basis corresponds to a spanning forest on the node set N in which each matrix B_i corresponds to a 1-tree.

A 1-tree on a node set N_i with cardinality n_i consists of a spanning tree on the node set together with one additional arc, thus forming exactly one cycle. The cycle may correspond to a self loop as in (a). This is the main distinction between the generalized network flow problem and the pure network problem. Thus a data structure which represents a spanning forest should be able to distinguish between subsets of nodes. That is, distinguish between the various 1-trees in the basis. Further, the data structure should facilitate the three essential operations of a primal simplex pivot: (1) pricing, or the determination of an entering variable, (2) the ratio test

operation which finds the representation of the incoming variable with respect to the current basis and then finds the leaving variable and (3) the update of the dual variables to effect the pricing operation.

II. Data Structure

An examination of the requirements for effecting the primal simplex operations makes it clear that the pricing operation is trivial once given the dual variables at any iteration. The column update or the ratio test operations are more involved. The entering arc may connect two nodes in the same 1-tree, or it may have one node in one 1-tree and the other in another 1-tree. There are several possibilities for the leaving arc:

(a) The entering arc is in a 1-tree, and the leaving arc is in the loop of the 1-tree. Therefore, a new loop will be formed, and the previous loop will be broken.

(b) The entering arc is in a 1-tree, and the leaving arc is in the loop formed by the entering arc. Therefore, there is no new loop formed and no old loop broken. This is similar to a pure network pivot.

(c) The entering arc is in a 1-tree. The leaving arc is neither in the old loop nor in the new loop formed by the entering arc. Therefore, the old loop remains intact, and a new loop is formed. Thus, the old 1-tree is split into two separate 1-trees.

(d) The entering arc connects two 1-trees, and the leaving arc is a loop arc. Thus, the two 1 trees are coalesced into a single 1-tree.

(e) The entering arc connects two 1-trees, and the leaving arc is not a loop arc. Thus, after the pivot, there are still two 1-trees with the old loops intact.

In order to effect pivots of the type summarized in (a) - (e), it is important to distinguish between loop arcs of a 1-tree and to be able to

distinguish between the various 1-trees of a forest. In general, since the number of 1-trees in a basis can vary, a data structure should be able to effect changes in spanning forests efficiently.

The data structure used for storing the spanning forest corresponding to the basis of a generalized network problem consists of the following node-length arrays:

p(i) The predecessor of a node i,

b(i) The brother of a node i, (0 for loop nodes),

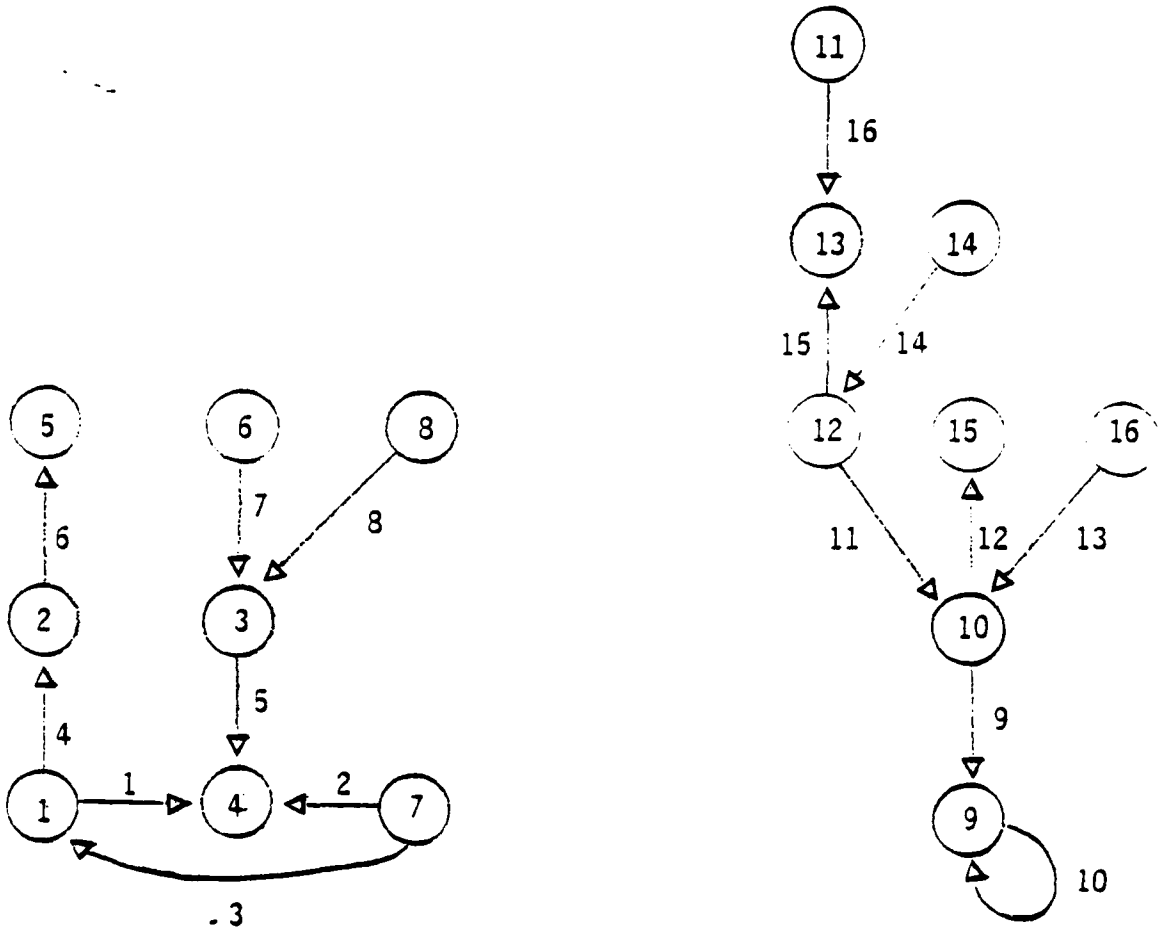
s(i) The successor of a node i,

l(i) The level of a node i. For a node which is not part of a loop, this gives the distance from the nearest node of the loop. For a node which is part of the loop, this is either 0 or a negative number. Zero for a self-loop. Negative for a loop node of a 1-tree where the absolute value gives the number of the particular 1-tree, i.e. the name given to the 1-tree.

a(i) The arc connecting node i to its predecessor. This number is positive if the arc is from the node to its predecessor and negative otherwise.

In Figure 1, this data structure is illustrated for a spanning forest on 16 nodes with two 1-trees. One of the 1-trees contains a self-loop while the other contains a loop with three arcs. The brother and successor labels allow a traversal of the 1-trees. While a preorder traversal [ ] has been shown to be suitable for a tree, it cannot determine the distinction between various subtrees of a 1-tree rooted on any one of the loop-nodes. A traversal of a 1-tree is possible using the brother and successor labels.

The spanning 1-tree can be regarded as several rooted trees together in a loop. If this loop is ignored, then every non-loop node has a unique path to the



Node	Predecessor	Brother	Successor	Level	Arc
1	4	0	2	-1	1
2	1	0	5	1	-4
3	4	0	6	1	5
4	7	0	3	-1	-2
5	2	0	0	2	-6
6	3	8	0	2	7
7	1	0	0	-1	3
8	3	0	0	2	8
9	9	0	10	0	10
10	9	0	12	1	9
11	13	0	0	4	16
12	10	15	13	2	11
13	12	14	11	3	-15
14	12	0	0	3	14
15	10	16	0	2	-12
16	10	0	0	2	13

Figure 1

nearest loop node which is its root. The predecessor of a non-loop node is the next node on the path to this root. For a loop node, the predecessor is the next node on the loop so that a trace of the loop can be effected by using the predecessor. For a self-loop, the predecessor of the root node is itself. Thus, the predecessor allows for a traversal from any node to the root and the entire loop. For example, in Figure 1, by making use of the predecessor, the traversal from node 5 down to node 1 can be effected and further, the traversal around the loop from node 1 to node 4 to node 7 and back to node 1.

Apart from the above mentioned arrays, the following node length arrays are used in addition:

f(i) the flow on arc a(i),

v(i) the dual variable for node i,

y(i) the updated column representation coefficient.

Thus, a total of 8-node length arrays are used together with at most half-node length arrays for the loop factors. There are only four arc length arrays necessary for storing the arc data. These are the to node node(x), the multiplier or gain g(x), the cost c(x) and the capacity cap(x).

III. Ratio Test

The ratio test operation consists of evaluating the vector y such that

$$By = a_k$$

Since $a_k = d_k e_i - g_k e_j$, where e_i is the i^{th} unit vector, and because of the block diagonal structure of the basis, B, the non-zero entries of the vector a_k may occur in the same block of B or in two different blocks of B. In the primal simplex specialization for the pure network problem, the ratio test can be performed as the vector y is evaluated since the entries in the

y vector are either +1, -1, or 0. Thus, only a single pass of the relevant arcs in the basis which produce non-zero entries in the y vector is required. For the specialization of the primal simplex algorithm for the generalized network problem, it is desirable to evaluate the y vector in a manner such that the analogous number of passes is a minimum. Two cases arise immediately:

(1) The non-zero entries of the entering arc occur in two different blocks, i.e., two 1-trees.

(2) The non-zero entries of the entering arc occur in the same block of the basis, i.e., one 1-tree.

By viewing the operation as

$$By' + By'' = d_k e_i - g_k e_j ,$$

the essential operation to be performed is the solution of the following system of equations, the system of equations for a single block of the basis B. A single block corresponds to a spanning 1-tree.

For simplicity, we consider only the rows and columns of the block which correspond to non-zero entries in the y vector to be evaluated. The system of equations then takes the form:

$$\begin{bmatrix} d_1 & & & & & \\ g_1 & d_2 & & & & \\ & g_2 & d_3 & & & \\ & & \ddots & \ddots & & \\ & & & g_{p-1} & d_p & \cdots & g_q \\ & & & & g_p & \ddots & \\ & & & & & & d_q \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_p \\ \\ y_q \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

Note that one of the entries in each column, either d_i or g_i will be a +1, and all the entries d_i are non-zero.

This corresponds to the graph shown in the figure 2. The incidence node for the entering arc corresponds to the node for row 1 or the above system. We shall assume that the nodes are numbered in the order of the rows. There are p-1 nodes until the first loop node, node p, is reached, and nodes p through q are all in the loop. Note that if g_q is zero, then the node q has a self-loop, and p = q. The solution to the above system is obtained as:

$$y_1 = 1/d_1$$
$$y_i = y_{i-1} \cdot (-g_{i-1}/d_i) \qquad i = 2,3,\ldots,p-1,p+1,\ldots,q$$
$$y_p = y_{p-1} \cdot [-g_{p-1}/d_p - G)]$$

where $F = (-g_p)\ldots(-g_q) / (d_p\ldots d_q)$ and $G = 1 - F$, and for self-loop $F = 0$, F is called the loop factor. This calculation can be performed on the 1-tree by making use of the suggested data structure. Tracing the path down to the loop is effected by making use of the predecessor index. The first node on the loop is identified by making use of the level index. Note that the loop factor F, as well as G, can be obtained when the 1-tree is constructed. Thus, it is not necessary to trace the loop twice to effect the above solution of equations. The y vector corresponding to any entering arc can be obtained by evaluating y' and y" from

$$By' = d_k e_i$$

$$By'' = g_k e_j$$

and then setting $y = d_k y' + g_k y''$.

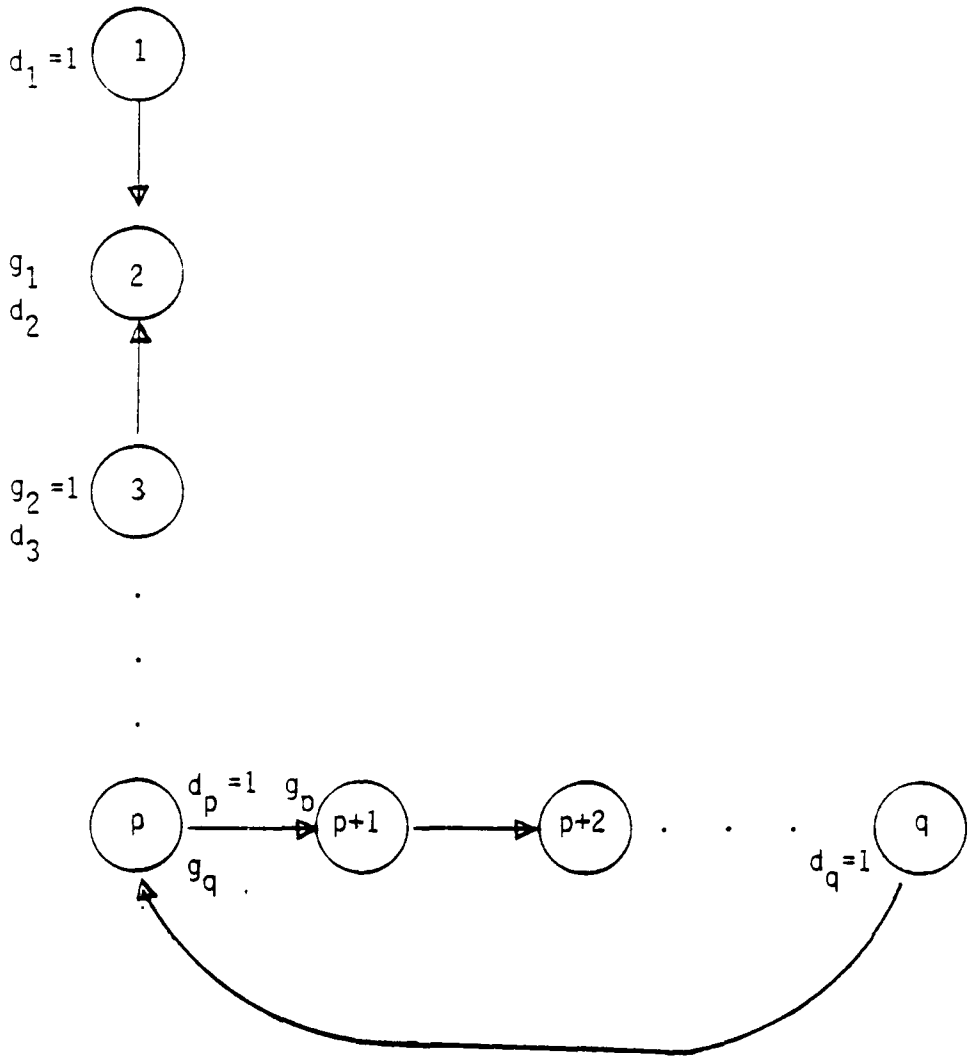


Figure 2

Orientation determines whether d_i or g_i is a +1.

Depending on whether the entering arc is incident to two blocks or one block the ratio test and the y vector evaluation can be performed simultaneously or simultaneously in part only. If there are two blocks then, the ratio test can be performed as the vectors y' and y" are evaluated. This is because no basic arc can appear in both y' and y". Thus no superfluous calculation is performed.

If there is only one block then, arcs on the path from node i to the loop and arcs on the path from node j to the loop may in fact overlap. It is desirable to arrange the calculations so that the ratio test can be performed as the y vector is evaluated. In order to perform this operation efficiently we define a node which is called the junction of the paths to the root. The junction is defined to be the common node with highest level on the two paths. It may not always exist. The following algorithm summarizes the ratio test operation.

Algorithm 1. Ratio Test

0. [Initialize]

   a. [set y] $y(x) := 0 \quad x = 1, 2, \ldots, m$

   b. [set junction flag] junction := 0

   c. [increase or decrease flow on arc k] $\delta := \text{sign}(\text{cap}(k))$

   d. [set leaving arc] $\mu := 0$

   e. [set Δ to bound] $\Delta := \| \text{cap}(k) \|$

1. Travel from node i to its root node, where the root node is the loop node that has the nearest distance from the node.

   a. [initialize] $T := 1$, $x := i$

   b. [root node ?] if $l(x) \leq 0$, then $T1 := T$

      [store the root node of i] $R1 := x$

      go to 2

Algorithm 1, continued.

c. [evaluate $y(x)$]

if $a(x) > 0$, then $y(x) := T + y(x)$

$T := -T \cdot g[a(x)]$

go to d.

otherwise $T := T/g[a(x)]$

$y(x) := T + y(x)$

$T := -T$

go to d.

d. Determine maximum allowable flow change

d.1. [increase flow ?] if $\delta \cdot y(x) > 0$ then go to d.3

d.2. [increase flow] if $\dfrac{cap[\,\|\,a(x)\,\|\,] - f(x)}{-\delta \cdot y(x)} \geq \Delta$

then go to d.4.

otherwise $\Delta := \dfrac{cap[\,\|\,(a(x))\,\|\,] - f(x)}{-\delta \cdot y(x)}$

$\mu := x$

go to d.4.

d.3. [decrease flow] if $\dfrac{f(x)}{\delta \cdot y(x)} \geq \Delta$

then go to d.4.

otherwise $\Delta := f(x)/(\delta \cdot y(x))$

$\mu := x$

$L_\mu := 1$

go to d.4.

d.4. [move to next node] $x := p(x)$

d.5. go to b.

Algorithm 1, continued.

2. Travel from node j to its root node

a. [initialize] T := g(k)

x := j

$\mu 1 := \mu$, $\mu := 0$

b. [root node ?] if $l(x) \leq 0$ then T2 := T

R2 := x

go to 3.

c. [junction node?] if junction = 0 and $y(x) \neq 0$

then junction = x ,

go to e.

d. [evaluate y(x) , determine maximum allowable flow change]

the same as 1.c and 1.d , except change $L_\mu := 1$ to $L_\mu \quad 2$

e. [redo ratio test along the path from node i to node junction ?]

if $\mu_1 > 0$ and $l(\text{junction}) \geq l(\mu_1)$

then go to f.

otherwise go to d.

f. [redo ratio test from i to junction]

f.1. [initialize] x := i

f.2. [finish ?] if x = junction then go to d.

f.3. [ratio test] the same as 1.d.1 - 1.d.4.

f.4. go to f.2

3. Travel along the loop

a. [i and j in the same subtree ?] if R1 = R2 then go to g.

b. [i and j in the same 1-tree ?]

if l(R1) = l(R2) ≠ 0 then go to f.

Algorithm 1, continued.

c. [evaluate y(x) along R1's loop[ ]

c.1 [initialize] x := R1

w := R1

T := T1

c.2. [self-loop ?] if l(w) = 0 then y(w) := T

go to d.

c.3. [loop factor] if a(w) > 0

then T := T/[1 - factor (- l(w))]

y(w) = y(w) + T

T := -T · (g(w))

go to d.

otherwise T := T/{[1-Factor(-l(w))]xg[-a(w)]}

y(w) = y(w) + T

T := -T

go to d.

c.4. [finish ?] if p(x) = w then go to e.

c.5. x: = p(x)

c.6. [evaluate y(x)] the same as 1.c.

d. [ratio test]

d.1. the same as 1.d.1 - 1.d.4.

d.2. go to c.4.

e. [evaluate y(x) along R2's loop node and do ratio test]

e.1. [initialize] x := R2

w := R2

T := T2

Algorithm 1, continued.

e.2. the same as c.2 - c.6 ; d.1, d.2, except

changing c.5 go to e to c.5 terminate ,

and changing $L_\mu = 1$ to $L_\mu = 2$

f. the node i, j in the same 1-tree but have different root node

[start from R1 travel the loop, but only y(x) is evaluated]

the same as c.1. - c.6. except changing "go to d" to

"go to c.4"

g. [combine two travel as one]

g.1. [initialize] x := R2

w := R2

T := T1 + T2

g.2. the same as e.2.

IV. Pivot and Dual Variable Update.

In order to obtain the proper dual variables and level indices, a subtree needs to be traversed. This traversal is effected by making use of the successor and brother labels. To traverse the entire subtree, the following rule is employed:

(a) If the node has a successor, visit it; otherwise

(b) If the node has a brother, visit the brother, otherwise

(c) If the predecessor of the node is the root of the subtree, terminate; otherwise, visit it.

The above is the essential traversal which is used. The particular pivot operation and the dual variable update dependent on the particular type of pivot to be performed. If the entering arc is in the same 1-tree of the forest then the operation is a little more complex than the case in which the two incident nodes of the entering arc are in different 1-trees.

At the termination of the ratio test algorithm, the maximum allowable change Δ is given, and if $\mu = 0$, then the entering arc is also the leaving arc. In this case only a flow change is required. Otherwise, the leaving arc is μ, which is on the path L_μ. The path from the tail node of the entering arc to its root node is considered to be path 1 and the other path as path 2.

The various possibilities in terms of the type of change in the basis occurring due to the specifics of the entering arc and the leaving arc have been summarized in the introduction. The whole operation is separated into three parts for exposition: Flow Update, Forest Update and the Dual Variable Update.

The flow update is rather straightforward and is summarized in Algorithm 2.

Algorithm 2. Flow Change

1. [have junction node?] if junction = 0 then go to 5
2. [update flow along path from i to junction]
   a. [initialize] x: = i
   b. [finish?] if x = junction then go to 3
   c. [update flow] f(x): = f(x) - $\delta \cdot \Delta \cdot y(x)$
   d. [next node] x: = p(x) , go to b
3. [update flow from j to its root]
   a. [initialize] x: = j
   b. [finish?] if x = R2 , then go to 4
   c. [update flow] f(x): = f(x) - $\delta \cdot \Delta \cdot y(x)$
   d. [next node] x: = p(x)
      go to b.
4. [update flow along the loop]
   a. [initialize] x: = R2 , w: = R2
   b. [update flow] f(x): = f(x) - $\delta \cdot \Delta \cdot y(x)$
   c. [finish?] if p(x) $\neq$ w then x: = p(x)
      go to b.
   d. terminate
5. [update flow from i to its root]
   a. [initialize] x: = i
   b. [finish ?] if x = R1 then go to 6
   c. [update flow] f(x): = f(x) - $\delta \cdot \Delta \cdot y(x)$
   d. [next node] x: = p(x)
      go to b.
6. [How many 1 - tree?] if l(R1) = l(R2) $\neq$ 0 or R1 = R2
   then go to 3.

Algorithm 2, continued.

7. [update flow along the loop of the 1 - tree which contain node i]

   a. [initialize] x: = R1 , w: = R1

   b.c. the same as 4.b and 4.c

   d. go to 3

In order to update the forest, Algorithm 3 is effected. This procedure makes use of the information obtained during the ratio test which determines the kind of update required. That is, the leaving arc is associated with node μ and L_μ indicates the path which contains node μ. The entering arc k is from node i to node j.

Algorithm 3. Update Predecessor orientation

1. [Initialize] if $L_\mu = 1$ then x: = i , w: = j , a_0: = k

   otherwise : = j , w: = i , a_0: = -k

2. [store some old data related to x]

   p_0: = p(x)

   $\bar{a}$: = -a(x)

   f_0: = f(x)

3. [delete x from its old family]

   a. [is x the successor of p_0?]

      if $x = s(p_0)$ then $s(p_0)$: = b(x)

      go to 4

      otherwise z: = $s(p_0)$

   b. [is x the brother of z?]

      if x = b(z) then b(z): = b(x)

      go to 4

   c. [check next one] z: = b(z) , go to b.

4. [set w be the predecessor of x] p(x): = w

Algorithm 3, continued.

5. [set x be the successor of y] $b(x) := s(w)$

$s(w) := x$

$f(x) := \Delta$

$a(x) := a_o$

6. [finish?] if $x = \mu$ then terminate

7. [next] $w := x$

$x := p_o$

$a_o := \overline{a}$

$\Delta := f_o$

go to 2.

Note that if the leaving arc is a loop arc, the pivot will break the old loop. For such a pivot, Algorithm 3 must be preceded by making the loop node a successor of its predecessor. The update of the forest is summarized in Algorithm 4.

Algorithm 4. Forest Update

1. [set flag for new loop] New: = 0

2. [how many 1 - tree?] if $l(R1) = l(R2) \neq 0$ or $R1 = R2$ then go to 4

3. [two 1 - tree case, no new loop formed]

   a. [old loop broken?] if $l(\mu) > 0$ then go to c

   b. [make the loop node be the successor of its predecessor]

      b.1. [initialize] $x := \mu$

      $w := p(x)$

      b.2. [make x be the w's successor]

      $b(x) := s(w)$,

      $s(w) := x$

Algorithm 4, continued.

b.3. [finish?] if w = μ then go to c

otherwise x: = w

w: = p(w)

go to b.2.

c. Use Algorithm 3.

d. terminate.

4. Only one 1 - tree, a new loop might be formed

a. [a new loop?]

if junction > 0 and $l(\mu) \leq l(\text{junction})$

then New = 1

and if $l(i) < l(j)$ then $L_\mu = 1$

otherwise $L_\mu = 2$

b. [old loop broken?] if $l(\mu) > 0$ then go to e

c. [old loop broken, new loop must be built] New = 1

d. [make the loop node be the successor]

the same as 3.b.

e. Use Algorithm 3.

f. if New = 0 then terminate

5. Delete the node on the new loop from its predecessor's family

a. [initialize] x: = i , w: = p(x)

b. [delete x from w's family]

b.1. if x = s(w) then s(w): = b(x)

go to c

otherwise w = s(w)

b.2. if x = b(w) then b(w): = b(x)

go to c

Algorithm 4, continued.

b.3. w: = b(w) , go to b.2.

c. [finish?] if i ≠ p(x) then x: = p(x)

w: = p(x)

go to b.

d. terminate.

Specifics of the update of the dual variables and the level index are summarized in Algorithm 5. Recall that the dual variables are the solution to the following system of equations:

$$v'B = c'$$

The dual variables are updated since the current basis and the updated basis differ in only one column. Only the following subsystem need be considered:

$$(v_1,\ldots,v_q) \begin{bmatrix} d_1 & & & & & \\ g_1 & d_2 & & & & \\ & \ddots & \ddots & & & \\ & & g_{p-1} & d_p & & \\ & & & g_p & \ddots & g_q \\ & & & & g_{q-1} & d_q \end{bmatrix} = (c_1,\ldots,c_q)$$

Thus, $v_k d_k + v_{k+1} g_k = c_k \qquad k = 1,\ldots,q-1$

$$v_q d_q + v_p g_q = c_q$$

The submatrix $\begin{bmatrix} d_p & & g_q \\ & \ddots & \\ g_p & & d_q \end{bmatrix}$ corresponds to the loop of 1-tree.

If there no new loops formed, the above submatrix remains unchanged; therefore $(v_p,\ldots v_q)$ satisfy the last p-q+1 equation of the above system, and other $(v_1,\ldots v_{p-1})$ can be easy calculated by the recursion equation traveling up the subtree. Hence the key point is when a new loop is formed, we should solve

$$(v_p,\ldots,v_q)\begin{bmatrix} d_p & & g_q \\ & \ddots & \\ g_p & & d_q \end{bmatrix} = (c_1,\ldots,c_q)$$

or $$\bar{v}^T \bar{B} = \bar{c}^T$$

$$\bar{v}^T = \bar{c}^T \bar{B}^{-1}$$

Introduce $$\bar{e} = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \qquad \bar{r} = \begin{bmatrix} r_p \\ \vdots \\ r_q \end{bmatrix}$$ such that

$$\bar{B}\,\bar{r} = \bar{e}$$

or $$\bar{r} = \bar{B}^{-1}\,\bar{e}$$

We already know how to get $\bar{r}$ from the $\bar{B}$ and $\bar{e}$.

However, $$v_p = \bar{v}^T \bar{e} = \bar{c}^T \bar{B}^{-1} \bar{e} = \bar{c}^T \bar{r}\ .$$

Based on the above analysis we have the following algorithm.

Algorithm 5. Update Dual Variable and Level Index

1. [any new loop ?] if new = 1 then go to 3.

Algorithm 5, continued.

2. at least one of node i or j has correct dual variable and level index

   a. [initialize] if $L_\mu = 1$ then w := j , x := j

   b. [check successor] d := x

      x := s(z), if x = 0 then go to e.

   c. [direction ?] if a(x) > 0, then v(x) := c(a(x)) - g[a(x)] · v(z)

      otherwise v(x) := [c(-a(x)) - v(z)]/g[-a(x)]

   d. [update level] l(x) = l(z) + 1 go to b.

   e. [finish ?] if z = w then go to g.

   f. [check brother] x := b(z) , z = p(z)

      if $x \neq 0$ then go to c.

      otherwise go to e.

   g. terminate.

3. evaluate the dual variable of new loop node i, assume the name of new loop is α .

   a. [initialize] x := i

      v_o := 0

      T := 1

      F := 1

   b. if z(x) > 0 then $v_o := v_o + T \cdot c(x)$

      T := -T · g(x)

      F := -g(x) · F

      otherwise T := T/g(x)

      $v_o := v_o + T \cdot c(x)$

      T := -T

      F := -F/g(x)

   c. [finish ?] if p(x) = i then go to d.

      otherwise x := p(x) , go to b.

Algorithm 5, continued

d. [store loop vector] Factor (α) := F

e. [store v(i)] $v(i) := v_0/(1-F)$

4. update dual variable and level by traveling entire new 1-tree.

a. [initialize] x := i go to c.

b. [set dual variable of the loop node x]

if $a(w) > 0$ then $v(x) := [c(a(w)) - v(w)]/g[a(w)]$

otherwise $v(x) := c(a(w)) - v(w) \cdot g(-a(w))$.

c. $l(x) := 0$, w := x.

d. [travel the subtree rooted at node x]

the same as 2.b. - 2.f.

g. [set level index for node w] $l(w) := -\alpha$

h. [finish ?] if $p(w) \neq i$ then x := p(w) , go to b.

otherwise terminate.

V. Computational Results

The data structures and algorithms presented in earlier sections have been implemented in FORTRAN to develop a generalized network code at the Center for Cybernetic Studies, The University of Texas at Austin. The computational testing for this code has been carried out on two sets of problems: Pure network problems whose specifications are given in Table 1, and Generalized Network problems whose specifications are given in Table 3. These problems have been generated using NETGEN [5] and one of its variants. The pure network problems were solved using a pure network code; this code and the generalized network code were both developed at the Center for Cybernetic Studies. The details of this computational testing are given in Table 2. This testing was carried out in order to determine the overhead required by the generalized network data structures over pure network data structures. Degradation in solution time varies from a factor of 2.2 to a factor of 4.0 while the degradation in time per pivot varies from a factor of 2.6 to 3.7. The results indicate that generalized network algorithms are on the order of 2.5 to 3.5 times slower than pure network algorithms.

The generalized network problems solved ranged from 200 nodes to 1000 nodes and 1500 arcs to 7000 arcs. The timings obtained for the solution of these problems are given in Table 4. The results indicate that the data structures presented are well-suited to carry out primal simplex operations for generalized network problems.

TABLE 1

PURE NETWORK PROBLEM SPECIFICATION

Problem	Node	Source	Sink	Arcs	Cost Range	Total Supply	Transshipment Sources	Transshipment Sink	% High Cost	% Arcs Capacitated	Upperbound Min	Upperbound Max	Random seed
1	400	8	60	1306	1-100	450000	0	0	30	20	16000	30000	13502460
2	400	8	60	2443	1-100	400000	0	0	30	20	16000	30000	13502460
3	400	8	60	1306	1-100	400000	0	0	30	20	20000	120000	13502460
4	400	8	60	2443	1-100	400000	0	0	30	20	20000	120000	13502460
5	400	8	60	1416	1-100	400000	5	50	30	40	16000	30000	13502460
6	400	8	60	2836	1-100	400000	5	50	30	40	16000	30000	13502460
7	400	8	60	1416	1-100	400000	5	50	30	40	20000	120000	13502460
8	400	8	60	2836	1-100	400000	5	50	30	40	20000	120000	13502460
9	400	4	12	1382	1-100	400000	0	0	30	80	16000	30000	13502460
10	400	4	12	2676	1-100	400000	0	0	30	80	16000	30000	13502460
11	400	4	12	1382	1-100	400000	0	0	30	80	20000	120000	13502460
12	400	4	12	2676	1-100	400000	0	0	30	80	20000	120000	13502460
13	1000	50	50	2900	1-100	1000000	0	0	0	0	0	0	13502460
14	1000	50	50	3400	1-100	1000000	0	0	0	0	0	0	13502460
15	1000	50	50	4800	1-100	1000000	0	0	0	0	0	0	13502460
16	1000	75	75	4385	1-100	1500000	0	0	0	0	0	0	13502460
17	1000	75	75	5107	1-100	1500000	0	0	0	0	0	0	13502460
18	1000	75	75	5730	1-100	1500000	0	0	0	0	0	0	13502460

TABLE 2

COMPUTATIONAL COMPARISON: PURE NETWORK PROBLEMS

	Pure Network Code		Generalized Code		Ratio	
Problem	Solve Time	# Iteration	Solve Time	# Iteration	On Solve Time	On Average Time Per Iteration
1	0.911	1224	2.346	1228	2.6	2.6
2	1.114	1621	3.319	1764	3.0	2.7
3	0.649	1006	2.017	1106	3.1	2.8
4	1.202	1652	3.562	1728	3.0	2.8
5	1.088	1411	3.069	1423	2.8	2.8
6	1.475	1947	3.837	1781	2.6	2.8
7	0.665	1045	2.473	1315	3.7	3.0
8	1.144	1611	2.863	1563	2.5	2.6
9	0.879	1540	3.510	1681	4.0	3.7
10	1.142	1707	3.002	1560	2.6	2.9
11	0.658	1263	1.456	964	2.2	2.9
12	1.004	1889	3.378	1740	3.4	3.7
13	3.562	3365	12.104	3609	3.4	3.2
14	3.090	2934	10.682	3341	3.5	3.0
15	4.402	4678	15.510	4445	3.5	3.7
16	5.902	4789	21.368	5181	3.6	3.3
17	7.826	5942	27.698	5839	3.5	3.6
18	8.204	6365	31.890	6864	3.9	3.6

TABLE 3

GENERALIZED PROBLEM SPECIFICATION

Problem	Node	Source	Sink	Transshipment Sources	Transshipment Sink	Arcs	Cost Range	Gain Range	Total Supply	Percent Capacitated	Bound Range	Random Seed
1	200	100	100	0	0	1500	1-100	.5-1.5	100000	0	--	13502460
2	200	100	100	0	0	2000	1-100	.5-1.5	100000	100	1000-2000	13502460
3	200	5	195	0	0	4000	1-100	.5-1.5	100000	100	1000-2000	13502460
4	200	15	50	10	20	6000	1-100	.25-.95	100000	100	1000-2000	13502460
5	300	150	150	0	0	1500	1-100	.5-1.5	100000	0	--	13502460
6	300	5	295	0	0	2000	1-100	.5-1.5	100000	0	--	13502460
7	300	135	165	135	130	4000	1-100	.5-1.5	100000	0	--	13502460
8	300	10	40	5	15	6000	1-100	.5-1.5	100000	0	--	13502460
9	399	2	50	1	20	7000	1-100	.5-1.5	100000	0	--	13502460
10	400	200	200	0	0	2000	1-100	.5-1.5	200	0	--	13502460
11	400	3	397	0	0	4000	1-100	.2-1.4	100000	0	--	13502460
12	400	20	100	5	50	5000	1-100	.3-1.7	100000	0	--	13502460
13	400	25	70	10	20	6000	1-100	.5-1.5	100000	100	1000-2000	13502460
14	400	30	50	0	0	7000	1-100	.5-1.5	100000	100	1000-2000	13502460
15	1000	20	100	5	20	6000	1-100	.4-1.4	200000	100	4000-6000	13502460
16	1000	20	50	20	50	6500	1-100	.5-1.5	200000	0	--	13502460
17	1000	30	400	10	30	7000	1-100	.7-1.2	200000	0	--	13502460

TABLE 4

COMPUTATIONAL RESULTS: GENERALIZED NETWORK PROBLEMS

TIMES ARE IN CPU SECONDS

2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
1318	570	2785	1479	820	2100	2286	3347	2059	632	2851	5933	6112	8482	6175	8873
4.685	2.365	9.805	6.476	4.859	10.950	6.079	7.954	11.438	5.534	11.149	27.606	30.046	53.787	27.03	76.304
200	200	200	300	300	300	300	399	400	400	400	400	400	1000	1000	1000
2006	970	6000	1523	1470	4000	6000	7000	2022	1188	5000	6000	5000	6000	6500	7000

REFERENCES

[1] A. Charnes and W.W. Cooper, Management Models and Industrial Applications of Linear Programming, Vols. I and II, John Wiley and Sons, Inc., New York New York, 1961.

[2] A. Charnes, J. Glover, K. Karney, K. Klingman, and J. Stutz, "Past, Present, and Future Development, Computational Efficiency, and Practical Use of Large-Scale Transportation and Transshipment Computer Codes," Computers and Operations Research, Vol.2, No. 2, pp. 71-81, 1975.

[3] F. Glover, J. Elam, and D. Klingman, "A Strongly Convergent Promal Algorithm for Generalized Networks," Center for Cybernetic Studies Report CCS 288, The University of Texas at Austin, Austin, Texas, 1977.

[4] F. Glover, D. Klingman, "On the Equivalence of Some Generalized Network Problems to Pure Network Problems," Mathematical Programming, 4.3, pp. 351-361, 1973.

[5] D. Klingman, A. Napier and J. Stutz, "Netgen: A Program for Generalizing Large Scale Capacitated Assignment, Transportation, and Minimum Cost Flow Network Problems," Management Science, Vol. 20, No. 5, January 1974.

[6] J.L. Kennington and R.V. Helgason, Algorithms for Network Programming, John Wiley and Sons, Inc., New York, New York, 1980.

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER CCS 483	2. GOVT ACCESSION NO. AD A165 242	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Design and Implementation of Data Structures for Generalized Networks		5. TYPE OF REPORT & PERIOD COVERED 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) I. Ali, A. Charnes, T. Song		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Center for Cybernetic Studies The University of Texas at Austin Austin, Texas 78712		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Office of Naval Research (Code 434) Washington, D.C.		12. REPORT DATE March 1984 13. NUMBER OF PAGES 31
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) Unclassified 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

This document has been approved for public release and sale; its distribution is unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Networks, Generalized Networks, Algorithms

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

The specialization of the simplex algorithm for the solution of generalized network flow problems rests on the fact that a basis for the problem may be represented graphically as a spanning forest in which each component is either a one-tree or a rooted tree. The design of a specialized algorithm for efficient solution of generalized network problems necessarily depends on data structures chosen to represent the basis. This paper presents the design and detailed algorithmic specification of the primal simplex algorithm for such problems. Computational testing to determine the overhead required by

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-014-6601

Unclassified
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

DTIC

FILMED

4-86

END